Amendments to the Specification:

Please replace the paragraph beginning on page 16, line 18, with the following amended paragraph:

For example, an IPv4 address can be thought of as a point on a number line from ~~0 to 232~~ 0 to 2^{32}. Each prefix can be thought of as an interval with two endpoints. The IP addresses that match a prefix P form an interval on this number line. For example, 10.*.*.* matches all IP addresses in the interval [10.0.0.0, 10.255.255.255]. In this way, a routing table can be looked upon as a set of intervals.

Please replace the paragraph beginning on page 17, line 10, with the following amended paragraph:

Finding the tiny-interval containing the lookup value involves finding the two consecutive ~~end-points~~ endpoints between which the lookup value lies. This is a classic search problem that can be solved using a comparison based multi-way tree such as B-tree. A B-tree is a hierarchical data structure that involves many levels of comparison in order to reach the correct leaf node.

Please replace the paragraph beginning on page 17, line 22, with the following amended paragraph:

In one embodiment, the number line is divided into T number of disjoint ranges so that each range gets about an equal number of ~~end-points~~ endpoints. Given a query point (e.g., IP address, character string, etc.), the interval between which two consecutive ~~end-points~~ endpoints it lies is identified. First, a range search is performed based on the T boundaries, to determine which partition it lies in. The boundaries of the range are typically referred to herein as partition points or a boundary prefix. This identified partition contains a small number of ~~end-points~~ endpoints.

Please replace the paragraph beginning on page 22, line 25, with the following amended paragraph:

FIG. 3C illustrates a process used in one embodiment to insert an endpoint in a partition. Note, the process may update one or more of the tiny trees within the identified partition. Processing begins with process block 340, and proceeds to process block 342 wherein the tiny tree and endpoint are identified. As determined in process block 345, if the partition is too ~~larch~~ large, then in process block 346, the partition is split into two partitions. In order to update the one or more data structures concurrently, new tiny trees are created and then afterwards, they will be linked into the active part of the one or more data structures. As such, two new tiny trees are created based on the existing endpoints plus the new endpoint. In process block 348, a back value is created for the split-off tiny tree, while the old back value typically remains the same for the old tiny tree. In process block 350, the range search entries (e.g., in the comparators or other partition identification mechanism) are updated to point to the new tiny trees with frees up the old tiny tree. Otherwise, in process block 360, a new tiny tree is created with the existing endpoints plus the new endpoint. In process block 362, the range search entry (e.g., in the comparators or other partition identification mechanism) is updated to point to the new tiny tree with frees up the old tiny tree.